Vivado Tutorial

Introduction

This tutorial guides you through the design flow using Xilinx Vivado software to create a simple digital circuit using Verilog HDL. A typical design flow consists of creating model(s), creating user constraint file(s), creating a Vivado project, importing the created models, assigning created constraint file(s), optionally running behavioral simulation, synthesizing the design, implementing the design, generating the bitstream, and finally verifying the functionality in the hardware by downloading the generated bitstream file. You will go through the typical design flow targeting the Artix-7 based Basys3 and Nexys4 DDR boards. The typical design flow is shown below. The circled number indicates the corresponding step in this tutorial.

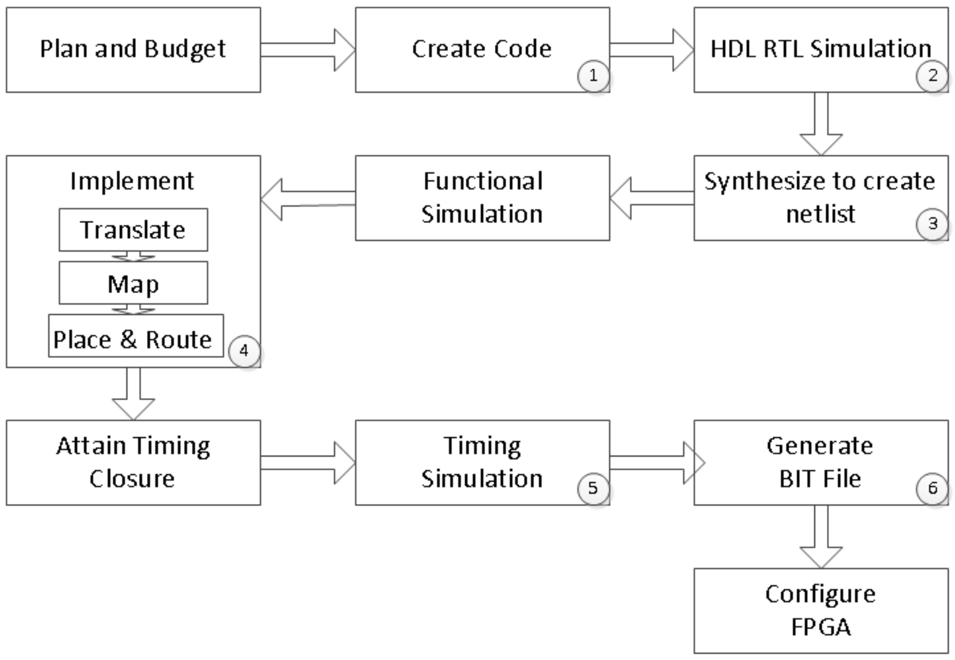


Figure 1. A typical design flow

Objectives

After completing this tutorial, you will be able to:

- Create a Vivado project sourcing HDL model(s) and targeting a specific FPGA device located on the Basys3 and Nexys4 DDR boards
- Use the provided partially completed Xilinx Design Constraint (XDC) file to constrain some of the pin locations
- Add additional constraints using the Tcl scripting feature of Vivado
- Simulate the design using the XSim simulator
- Synthesize and implement the design
- Generate the bitstream
- Configure the FPGA using the generated bitstream and verify the functionality

Procedure

This tutorial is broken into steps that consist of general overview statements providing information on the detailed instructions that follow. Follow these detailed instructions to progress through the tutorial.

Design Description

The design consists of some inputs directly connected to the corresponding output LEDs. Other inputs are logically operated on before the results are output on the remaining LEDs as shown in **Figure 2**.

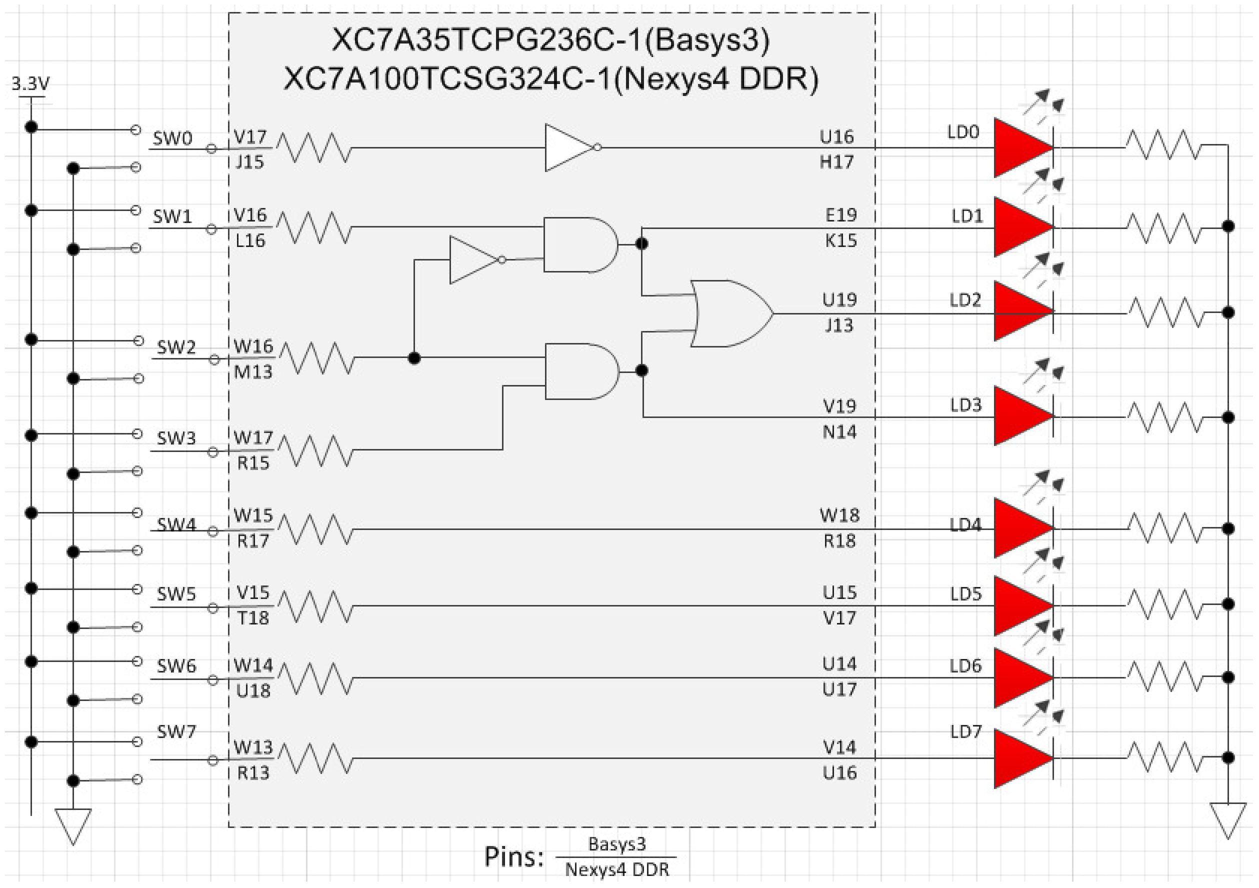


Figure 2. Completed Design

General Flow for this tutorial

- Create a Vivado project and analyze source files
- Simulate the design using XSim simulator
- Synthesize the design
- Implement the design
- Perform the timing simulation
- Verify the functionality in hardware using the Basys3 or Nexys4 DDR board

Create a Vivado Project using IDE Step 1

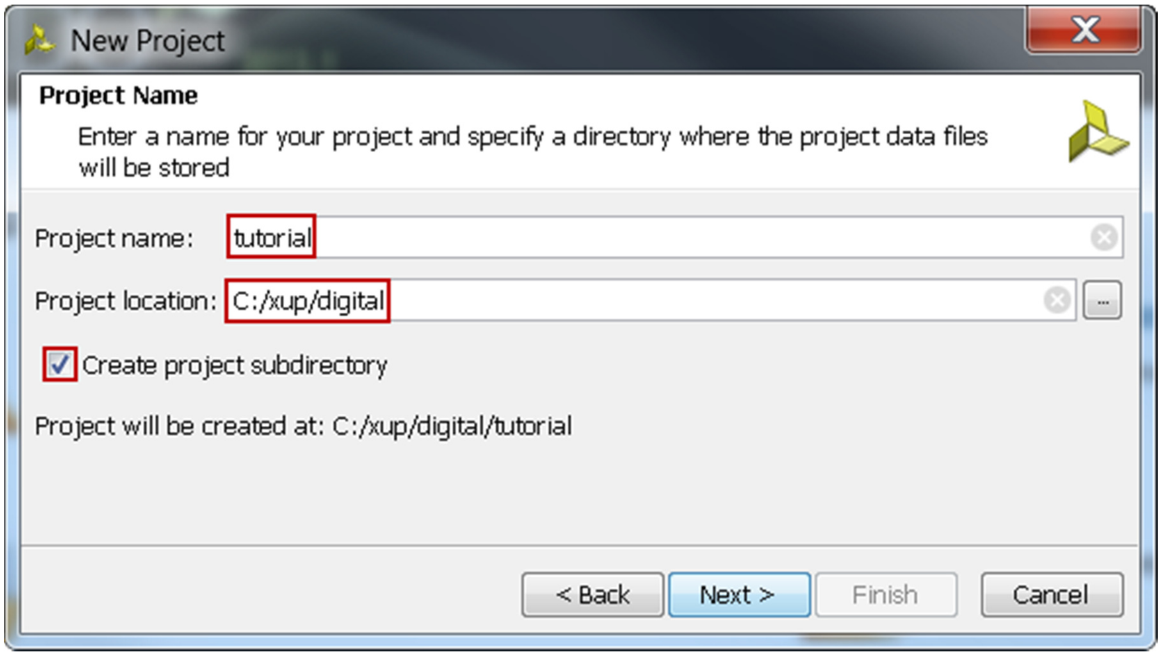
1-1. Launch Vivado and create a project targeting the xc7a35tcpg236-1 (Basys3) or xc7a100tcsg324-1 (Nexys4 DDR) device and using the Verilog HDL. Use the provided tutorial.v and Nexys4DDR_Master.xdc or Basys3_Master.xdc files from the *sources/tutorial* directory.

1-1-1. Open Vivado by selecting **Start > All Programs > Xilinx Design Tools > Vivado 2015.1 > Vivado 2015.1**

1-1-2. Click **Create New Project** to start the wizard. You will see *Create A New Vivado Project* dialog box. Click **Next**.

1-1-3. Click the Browse button of the *Project location* field of the **New Project** form, browse to **c:\xup\digital**, and click **Select**.

1-1-4. Enter **tutorial** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.


New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored

Project name: tutorial

Project location: C:/xup/digital

[✓] Create project subdirectory

Project will be created at: C:/xup/digital/tutorial

Figure 3. Project Name and Location entry

1-1-5. Select **RTL Project** option in the *Project Type* form, and click **Next**.

1-1-6. Select **Verilog** as the *Target language* and *Simulator language* in the *Add Sources* form.

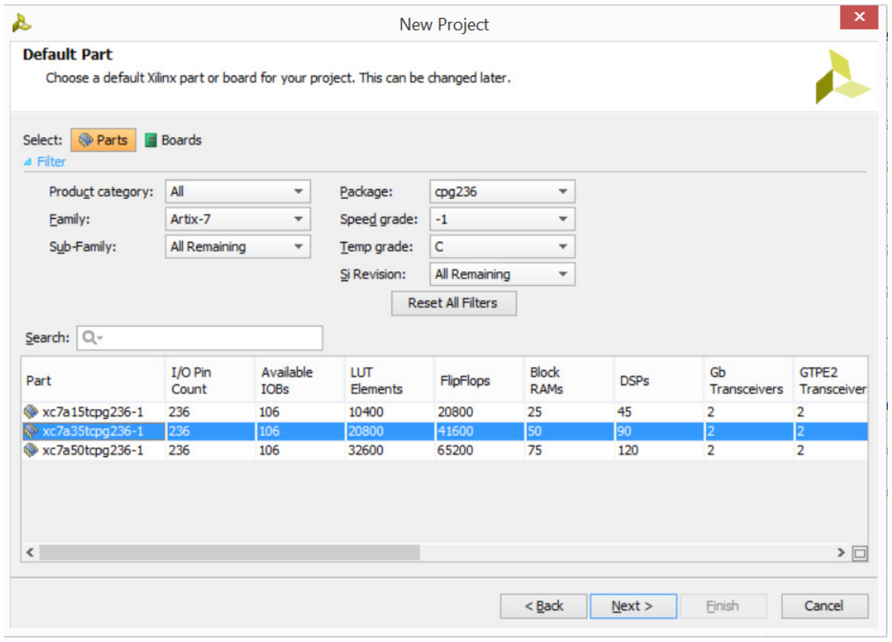
1-1-7. Click on the **Green Plus** button, then **Add Files…** button, browse to the **c:\xup\digital\sources\tutorial** directory, select *tutorial.v,* click **Open**, and verify the **Copy sources into project** box is checked, then click **Next**.

1-1-8. Click **Next** at the *Add Existing IP* form, since we do not have any pre-canned IP to be used in this design.

1-1-9. In the *Add Constraints* form, click on the **Green Plus** button, then the **Add Files…** button, browse to the **c:\xup\digital\sources\tutorial** directory, select *Basys3_Master.xdc* (for Basys3) or *Nexys4DDR_Master.xdc* (for Nexys4 DDR), click **Open**, and then click **Next.**

The XDC constraint file assigns the physical IO locations on FPGA to the switches and LEDs located on the board. This information can be obtained either through a board's schematic or board's user guide.

1-1-10. In the *Default Part* form, using the **Parts** option and various drop-down fields of the **Filter** section, select the **xc7a35tcpg236-1** part (for Basy3) or **xc7a100tcsg324-1** part (for Nexys4 DDR). Click **Next**.



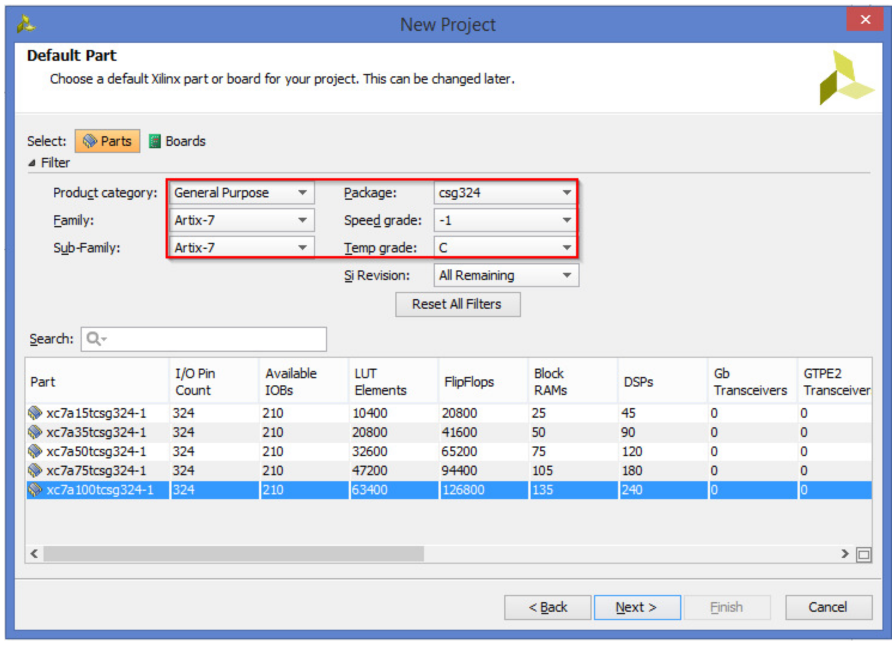
Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Product category:	All
Family:	Artix-7
Sub-Family:	All Remaining
Package:	cpg236
Speed grade:	-1
Temp grade:	C
Si Revision:	All Remaining

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GTPE2 Transceiver
xc7a15tcpg236-1	236	106	10400	20800	25	45	2	2
xc7a35tcpg236-1	236	106	20800	41600	50	90	2	2
xc7a50tcpg236-1	236	106	32600	65200	75	120	2	2

Figure 4. Part Selection for Basys3



Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Product category:	General Purpose
Family:	Artix-7
Sub-Family:	Artix-7
Package:	csg324
Speed grade:	-1
Temp grade:	C
Si Revision:	All Remaining

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GTPE2 Transceiver
xc7a15tcsg324-1	324	210	10400	20800	25	45	0	0
xc7a35tcsg324-1	324	210	20800	41600	50	90	0	0
xc7a50tcsg324-1	324	210	32600	65200	75	120	0	0
xc7a75tcsg324-1	324	210	47200	94400	105	180	0	0
xc7a100tcsg324-1	324	210	63400	126800	135	240	0	0

Figure 4. Part Selection for Nexys4 DDR

1-1-11. Click **Finish** to create the Vivado project.

Use the Windows Explorer and look at the **c:\xup\digital\tutorial** directory. You will see that the **tutorial.srcs** and other directories, and the **tutorial.xpr** (Vivado) project file have been created. Two sub-directories, **constrs_1** and **sources_1**, are created under the **tutorial.srcs** directory; deep down under them, the copied *Nexys4DDR_Master*.xdc or *Basys3_Master.xdc* (constraint) and *tutorial.v* (source) files respectively are placed.

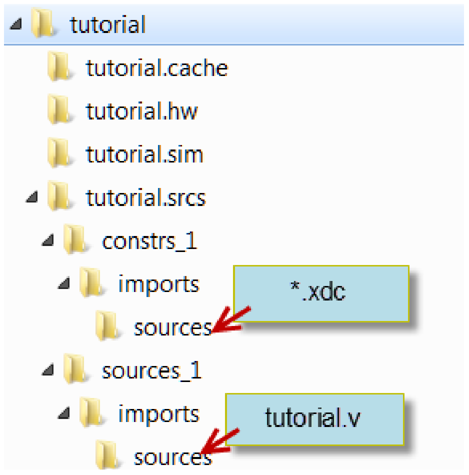


Figure 5. Generated directory structure

1-2. Open the tutorial.v source and analyze the content.

1-2-1. In the *Sources* pane, double-click the **tutorial.v** entry to open the file in text mode.

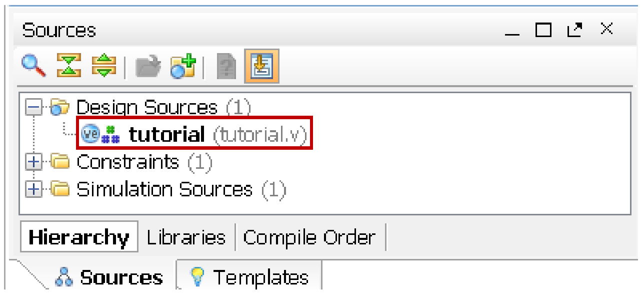


Figure 6. Opening the source file

1-2-2. Notice in the Verilog code that the first line defines the timescale directive for the simulator. Lines 2-5 are comment lines describing the module name and the purpose of the module.

1-2-3. Line 7 defines the beginning (marked with keyword **module**) and Line 19 defines the end of the module (marked with keyword **endmodule**).

1-2-4. Lines 8-9 define the input and output ports whereas lines 12-17 define the actual functionality.

1-3. Open the *Basys3_Master.xdc or Nexys4DDR_Master.xdc* source, analyze the content and edit the file.

1-3-1. In the *Sources* pane, expand the *Constraints* folder and double-click the ***Basys3_Master.xdc (Basys3) or Nexys4DDR_Master.xdc (Nexys4 DDR)*** entry to open the file in text mode.

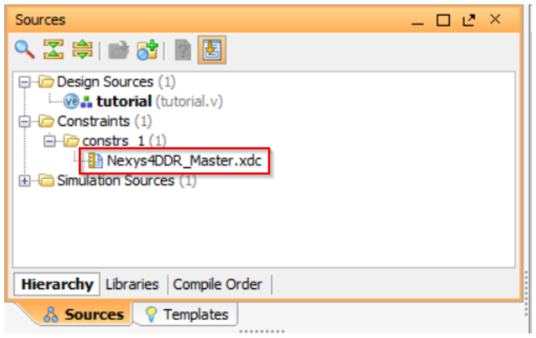
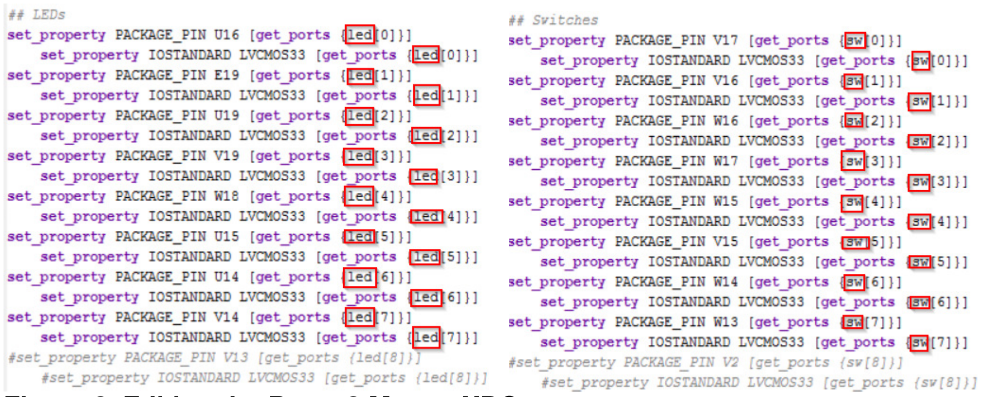


Figure 7. Opening the constraint file

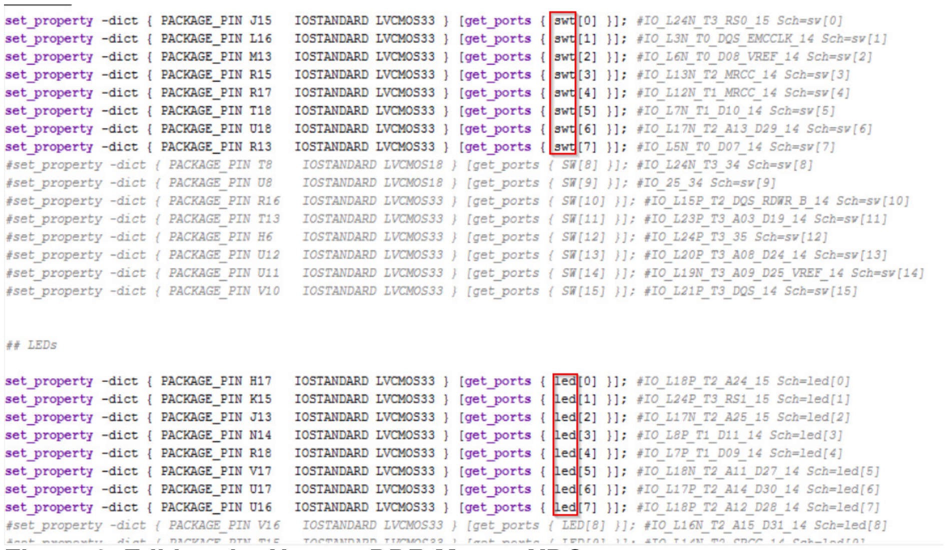
1-3-2. Uncomment the SW[7:0] by deleting the # sign or by highlighting SW[7:0] and pressing CRTL /. Uncomment LED[7:0]. The pin names will need to be changed to match the pin names in the *tutorial.v* file. Change *sw* to *swt* and *LED* to *led*.



```
## LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
#set_property PACKAGE_PIN V13 [get_ports {led[8]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
```

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
#set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
```

Figure 8. Editing the Basys3 Master XDC



```
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { swt[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { swt[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { swt[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { swt[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { swt[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { swt[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { swt[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { swt[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]


## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { led[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { led[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { led[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
```

Figure 8. Editing the Nexys4 DDR Master XDC

1-3-3. Change the sw[*] name to swt[*], and LED[*] to led[*] as the port names in the model are swt and led.

1-3-4. Close the *Basys3_Master.xdc* or the *Nexys4DDR_Master.xdc* file saving the changes.

1-4. Perform RTL analysis on the source file.

1-4-1. Expand the *Open Elaborated Design* entry under the *RTL Analysis* tasks of the *Flow Navigator* pane and click on **Schematic**.

1-4-2. Click **OK** to run the analysis.

The model (design) will be elaborated and a logic view of the design is displayed.

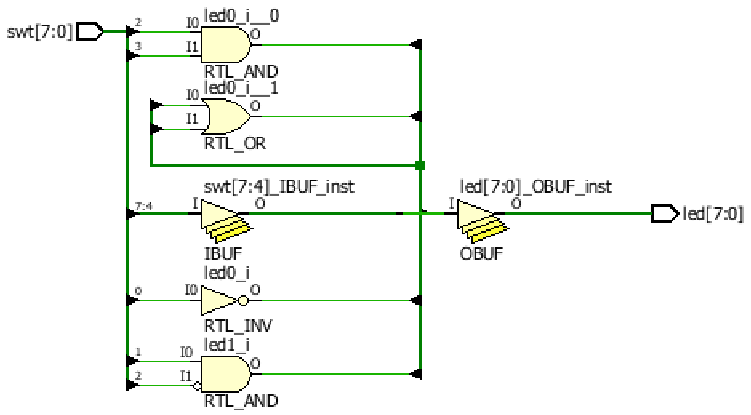


Figure 9. A logic view of the design

Notice that some of the switch inputs go through gates before being output to LEDs and the rest go straight through to LEDs as modeled in the file.

1-5. I/O constraints

1-5-1. Once RTL analysis is performed, another standard layout called the *I/O Planning* is available. Click on the drop-down button and select the *I/O Planning* layout.

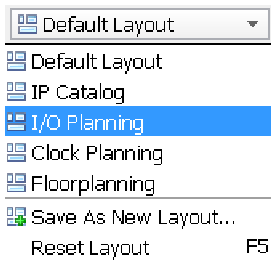
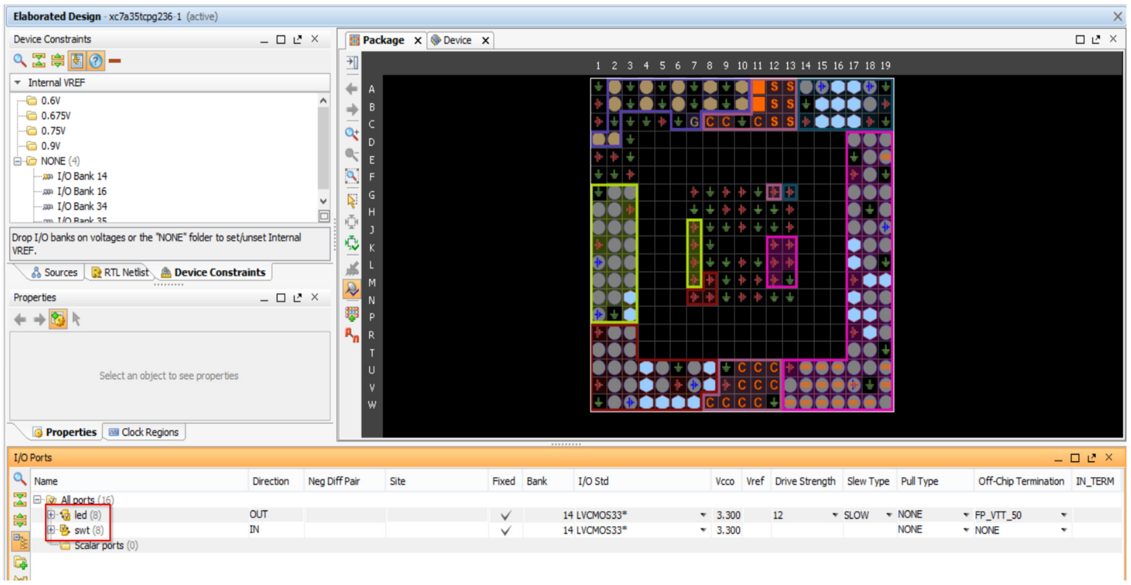


Figure 10. I/O Planning layout selection

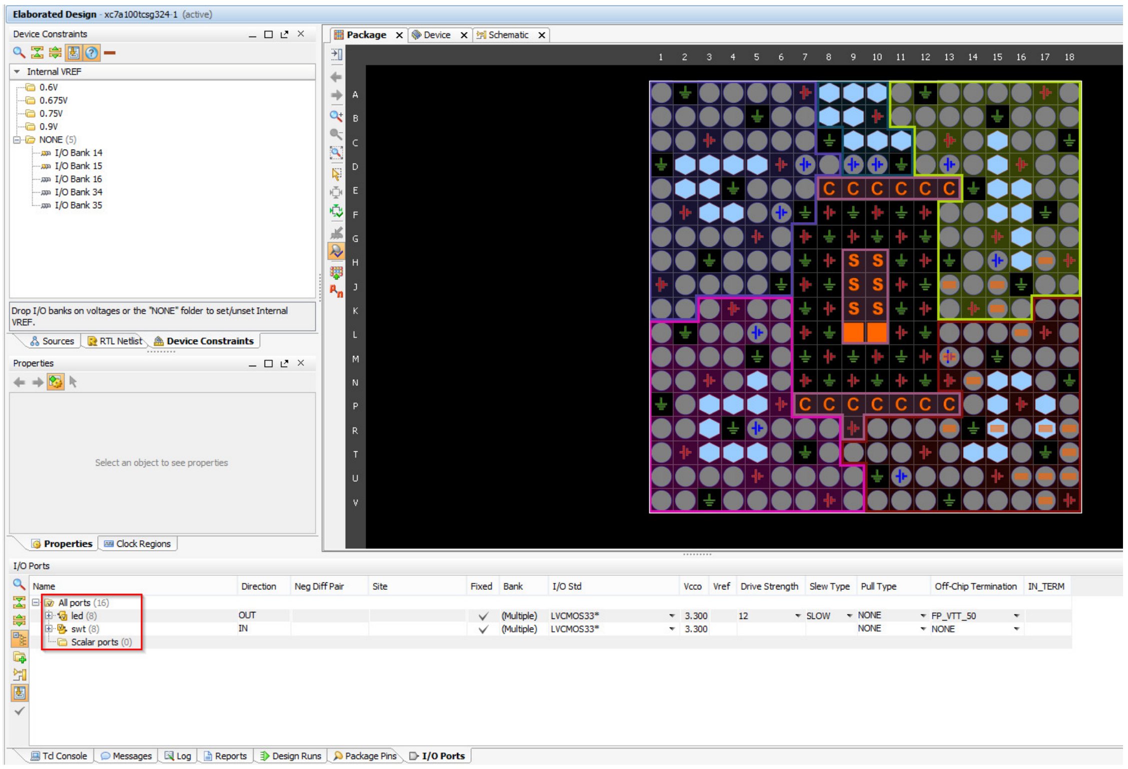
Notice that the Package view is displayed in the Auxiliary View area, RTL Netlist tab is selected, and I/O ports tab is displayed in the Console View area. Also notice that design ports (led and swt) are listed in the I/O Ports tab with both having multiple I/O standards.

Move the mouse cursor over the Package view, highlighting different pins. Notice the pin site number is shown at the bottom of the Vivado GUI, along with the pin type (User IO, GND, VCCO…) and the I/O bank it belongs to.



Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (16)													
led (8)	OUT			✓	14	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
swt (8)	IN			✓	14	LVCMOS33*	3.300				NONE	NONE	
Scalar ports (0)													

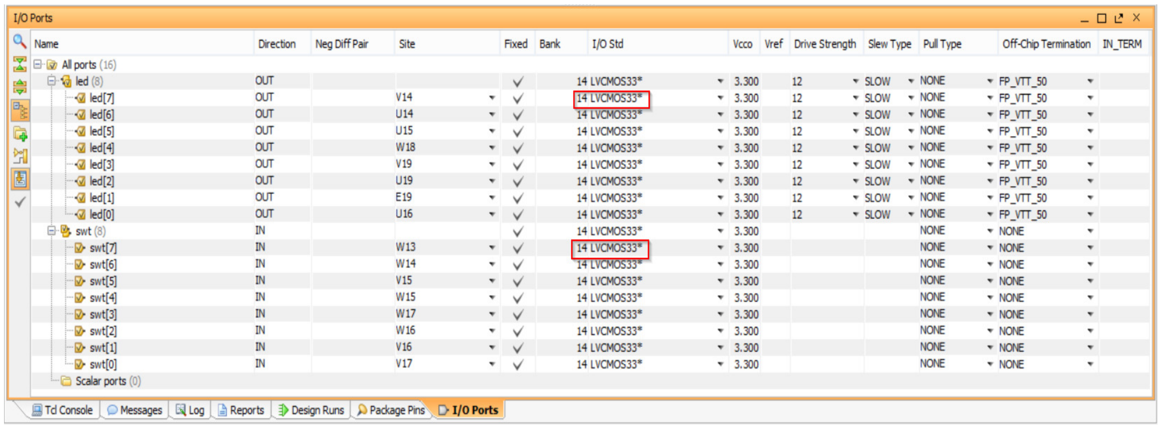
Figure 11. I/O Planning layout view for Basys3



Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (16)													
led (8)	OUT			✓	(Multiple)	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
swt (8)	IN			✓	(Multiple)	LVCMOS33*	3.300				NONE	NONE	
Scalar ports (0)													

Figure 11. I/O Planning layout view for Nexys4 DDR

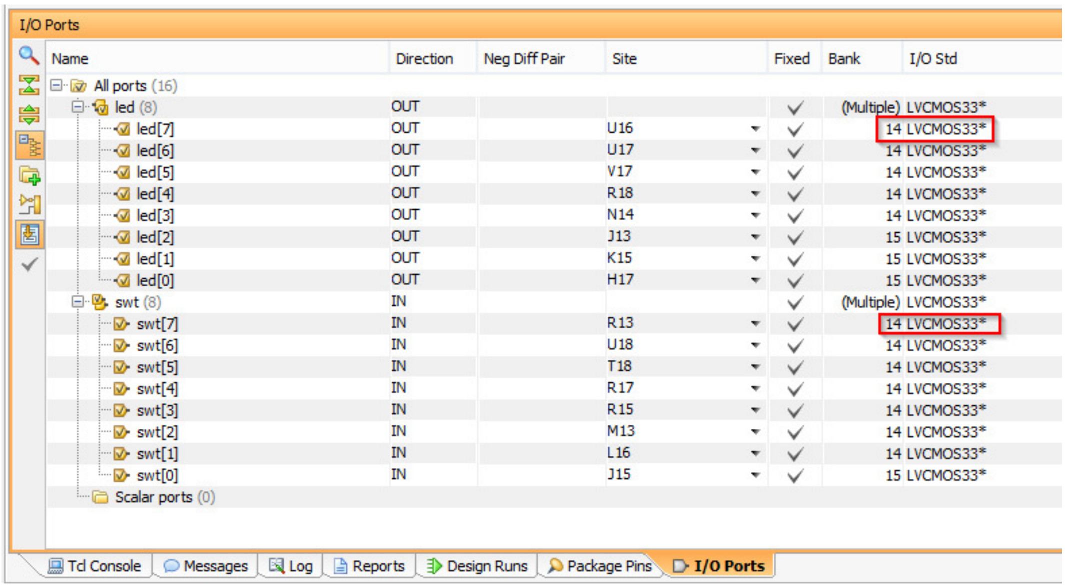
You can expand the **led** and **swt** ports by clicking on the + box and observe that led [7:0] and swt[7:0] have assigned pins and uses LVCMOS33 I/O standard. To change the I/O Standard, you would click in the **I/O Std** of the desired port and select the appropriate I/O Standard. The master XDC file already has the correct I/O Standard from editing the file in step 1-3-3.



I/O Ports

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (16)													
led (8)	OUT			✓	14	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
led[7]	OUT		V14	✓	14	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
led[6]	OUT		U14	✓	14	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
led[5]	OUT		U15	✓	14	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
led[4]	OUT		W18	✓	14	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
led[3]	OUT		V19	✓	14	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
led[2]	OUT		U19	✓	14	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
led[1]	OUT		E19	✓	14	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
led[0]	OUT		U16	✓	14	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
swt (8)	IN			✓	14	LVCMOS33*	3.300				NONE	NONE	
swt[7]	IN		W13	✓	14	LVCMOS33*	3.300				NONE	NONE	
swt[6]	IN		W14	✓	14	LVCMOS33*	3.300				NONE	NONE	
swt[5]	IN		V15	✓	14	LVCMOS33*	3.300				NONE	NONE	
swt[4]	IN		W15	✓	14	LVCMOS33*	3.300				NONE	NONE	
swt[3]	IN		W17	✓	14	LVCMOS33*	3.300				NONE	NONE	
swt[2]	IN		W16	✓	14	LVCMOS33*	3.300				NONE	NONE	
swt[1]	IN		V16	✓	14	LVCMOS33*	3.300				NONE	NONE	
swt[0]	IN		V17	✓	14	LVCMOS33*	3.300				NONE	NONE	
Scalar ports (0)													

Figure 12. I/O Ports tab for Basys3

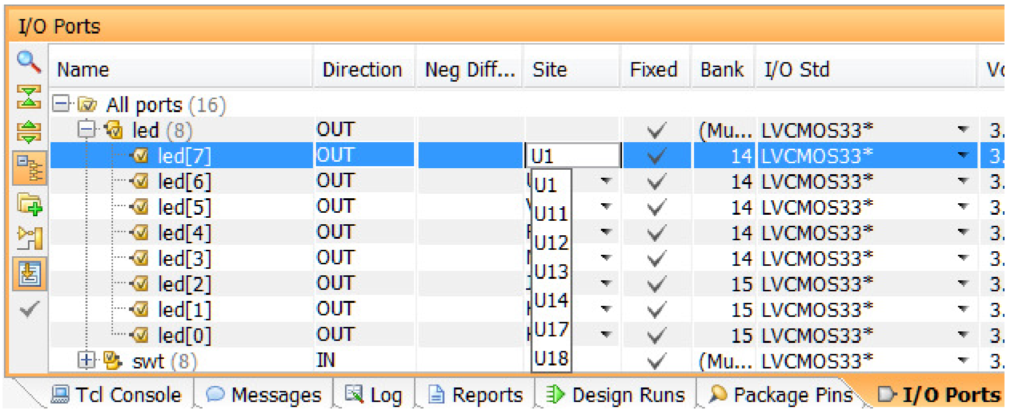


I/O Ports

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std
All ports (16)						
led (8)	OUT			✓	(Multiple)	LVCMOS33*
led[7]	OUT		U16	✓	14	LVCMOS33*
led[6]	OUT		U17	✓	14	LVCMOS33*
led[5]	OUT		V17	✓	14	LVCMOS33*
led[4]	OUT		R18	✓	14	LVCMOS33*
led[3]	OUT		N14	✓	14	LVCMOS33*
led[2]	OUT		J13	✓	15	LVCMOS33*
led[1]	OUT		K15	✓	15	LVCMOS33*
led[0]	OUT		H17	✓	15	LVCMOS33*
swt (8)	IN			✓	(Multiple)	LVCMOS33*
swt[7]	IN		R13	✓	14	LVCMOS33*
swt[6]	IN		U18	✓	14	LVCMOS33*
swt[5]	IN		T18	✓	14	LVCMOS33*
swt[4]	IN		R17	✓	14	LVCMOS33*
swt[3]	IN		R15	✓	14	LVCMOS33*
swt[2]	IN		M13	✓	14	LVCMOS33*
swt[1]	IN		L16	✓	14	LVCMOS33*
swt[0]	IN		J15	✓	15	LVCMOS33*
Scalar ports (0)						

Figure 12. I/O Ports tab for Nexys4 DDR

The ports are already assigned the pins. If you would like to assign pins in this view, you would click under the *Site* column across the desire port row to bring up a drop-down box. Type in the appropriate **Port Variable** to jump to the pins with that variable. Scroll-down until you see the correct port name, and then select it and hit the *Enter* key to assign the pin.



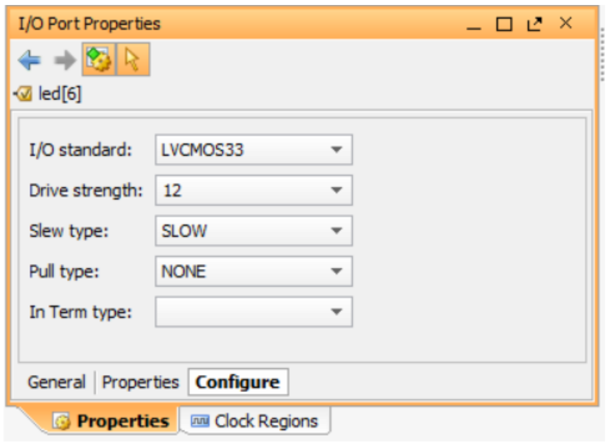
I/O Ports

Name	Direction	Neg Diff...	Site	Fixed	Bank	I/O Std	Vc
All ports (16)							
led (8)	OUT			✓	(Mu...	LVCMOS33*	3.
led[7]	OUT		U1	✓	14	LVCMOS33*	3.
led[6]	OUT			✓	14	LVCMOS33*	3.
led[5]	OUT			✓	14	LVCMOS33*	3.
led[4]	OUT			✓	14	LVCMOS33*	3.
led[3]	OUT			✓	14	LVCMOS33*	3.
led[2]	OUT			✓	15	LVCMOS33*	3.
led[1]	OUT			✓	15	LVCMOS33*	3.
led[0]	OUT			✓	15	LVCMOS33*	3.
swt (8)	IN			✓	(Mu...	LVCMOS33*	3.

Figure 13. Assigning pin location

You can also assign the pin by selecting its entry in the I/O ports tab, and dragging it to the Package view, and placing it at the desired location

You can also assign the I/O standard by selecting its entry in the I/O Ports tab, selecting the Configure tab of the I/O Port Properties window, followed by clicking the drop-down button of the I/O standard field, selecting LVCMOS33.



I/O standard:	LVCMOS33
Drive strength:	12
Slew type:	SLOW
Pull type:	NONE
In Term type:	

Figure 14. Assigning I/O standard through the I/O Port Properties form

You can also assign the pin constraints and I/O standards using tcl commands, by typing the command in the Tcl Console tab to assign the R13 pin location and the LVCSMOS33 I/O like shown below and hitting the Enter key after the command.

```
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports
{ swt[7] }];
```

Simulate the Design using the XSim Simulator Step 2

2-1. Add the tutorial_tb.v testbench file.

2-1-1. Click **Add Sources** under the *Project Manager* tasks of the *Flow Navigator* pane.

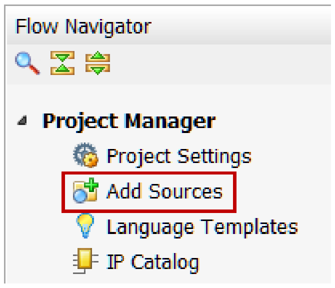
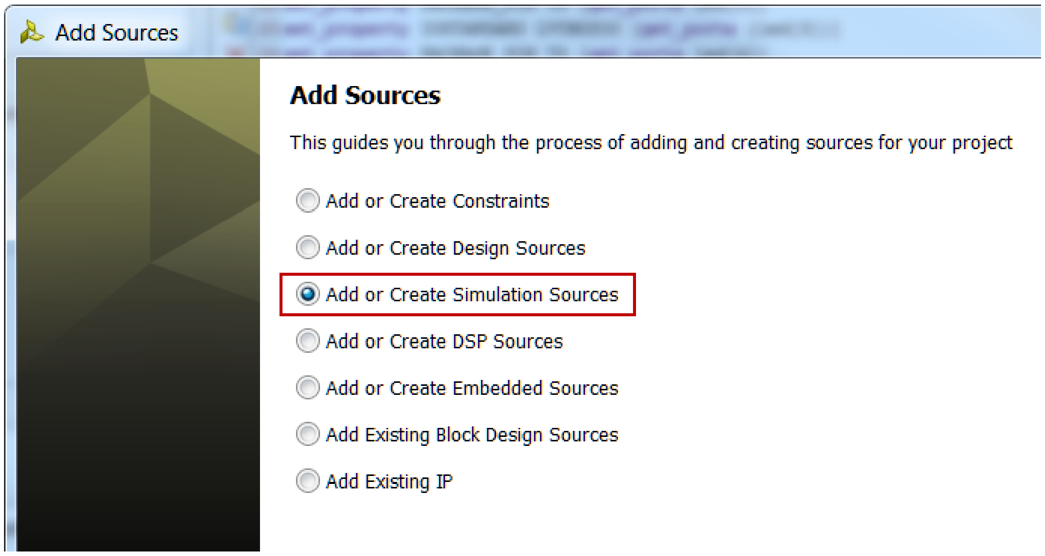


Figure 15. Add Sources

2-1-2. Select the *Add or Create Simulation Sources* option and click **Next**.



Add Sources

This guides you through the process of adding and creating sources for your project

- ○ Add or Create Constraints
- ○ Add or Create Design Sources
- [✓ Add or Create Simulation Sources]
- ○ Add or Create DSP Sources
- ○ Add or Create Embedded Sources
- ○ Add Existing Block Design Sources
- ○ Add Existing IP

Figure 16. Selecting Simulation Sources option

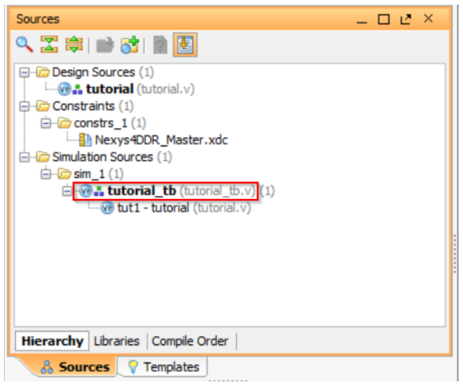
2-1-3. In the *Add Sources Files* form, click the **Green Plus** button then click the **Add Files…** button.

2-1-4. Browse to the **c:\xup\digital\sources\tutorial** folder and select *tutorial_tb.v* and click **OK**.

2-1-5. Click **Finish**.

2-1-6. Select the *Sources* tab and expand the *Simulation Sources* group.

The tutorial_tb.v file is added under the *Simulation Sources* group, and **tutorial.v** is automatically placed in its hierarchy as a tut1 instance.

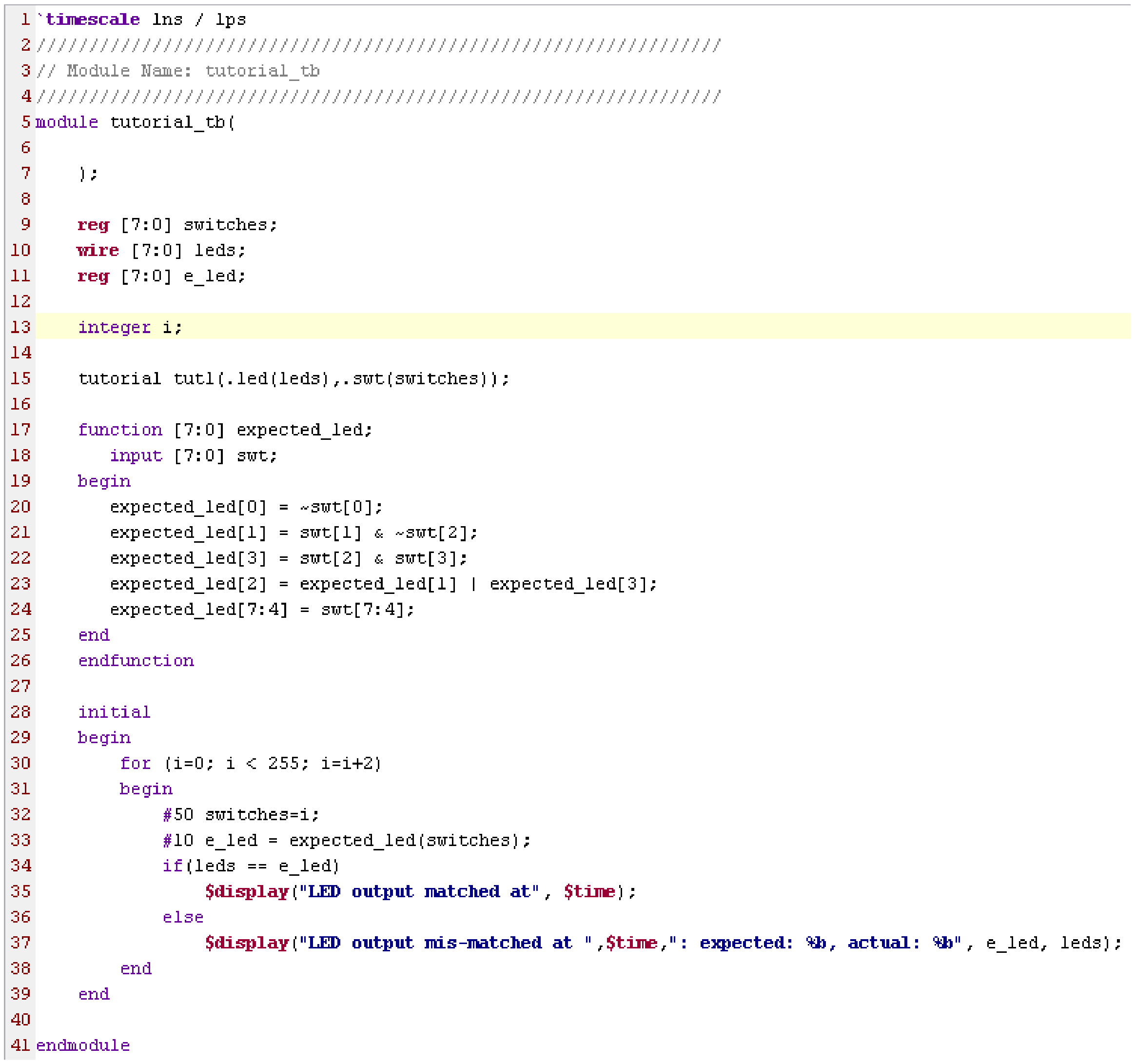


```
Design Sources (1)
    tutorial (tutorial.v)
Constraints (1)
    constrs_1 (1)
        Nexys4DDR_Master.xdc
Simulation Sources (1)
    sim_1 (1)
        tutorial_tb (tutorial_tb.v) (1)
            tut1 - tutorial (tutorial.v)
```

Figure 17. Simulation Sources hierarchy

2-1-7. Using the Windows Explorer, verify that the **sim_1** directory is created at the same level as constrs_1 and sources_1 directories under the tutorial.srcs directory, and that a copy of *tutorial_tb.v* is placed under **tutorial.srcs > sim_1 > imports > sources**.

2-1-8. Double-click on the **tutorial_tb** in the *Sources* pane to view its contents.



```
 1 `timescale 1ns / 1ps
 2 //////////////////////////////////////////////////////////////////////////
 3 // Module Name: tutorial_tb
 4 //////////////////////////////////////////////////////////////////////////
 5 module tutorial_tb(
 6 
 7     );
 8 
 9     reg [7:0] switches;
10     wire [7:0] leds;
11     reg [7:0] e_led;
12 
13     integer i;
14 
15     tutorial tut1(.led(leds),.swt(switches));
16 
17     function [7:0] expected_led;
18        input [7:0] swt;
19     begin
20        expected_led[0] = ~swt[0];
21        expected_led[1] = swt[1] & ~swt[2];
22        expected_led[3] = swt[2] & swt[3];
23        expected_led[2] = expected_led[1] | expected_led[3];
24        expected_led[7:4] = swt[7:4];
25     end
26     endfunction
27 
28     initial
29     begin
30         for (i=0; i < 255; i=i+2)
31         begin
32             #50 switches=i;
33             #10 e_led = expected_led(switches);
34             if(leds == e_led)
35                 $display("LED output matched at", $time);
36             else
37                 $display("LED output mis-matched at ",$time,": expected: %b, actual: %b", e_led, leds);
38         end
39     end
40 
41 endmodule
```

Figure 18. The self-checking testbench

The testbench defines the simulation step size and the resolution in line 1. The testbench module definition begins on line 5. Line 15 instantiates the DUT (device/module under test). Lines 17 through 26 define the same module functionality for the expected value computation. Lines 28 through 39 define the stimuli generation and compares the expected output with what the DUP provides. Line 41 ends the testbench. The $display task will print the message in the simulator console window when the simulation is run.

2-2. Simulate the design for 200 ns using the XSim simulator.

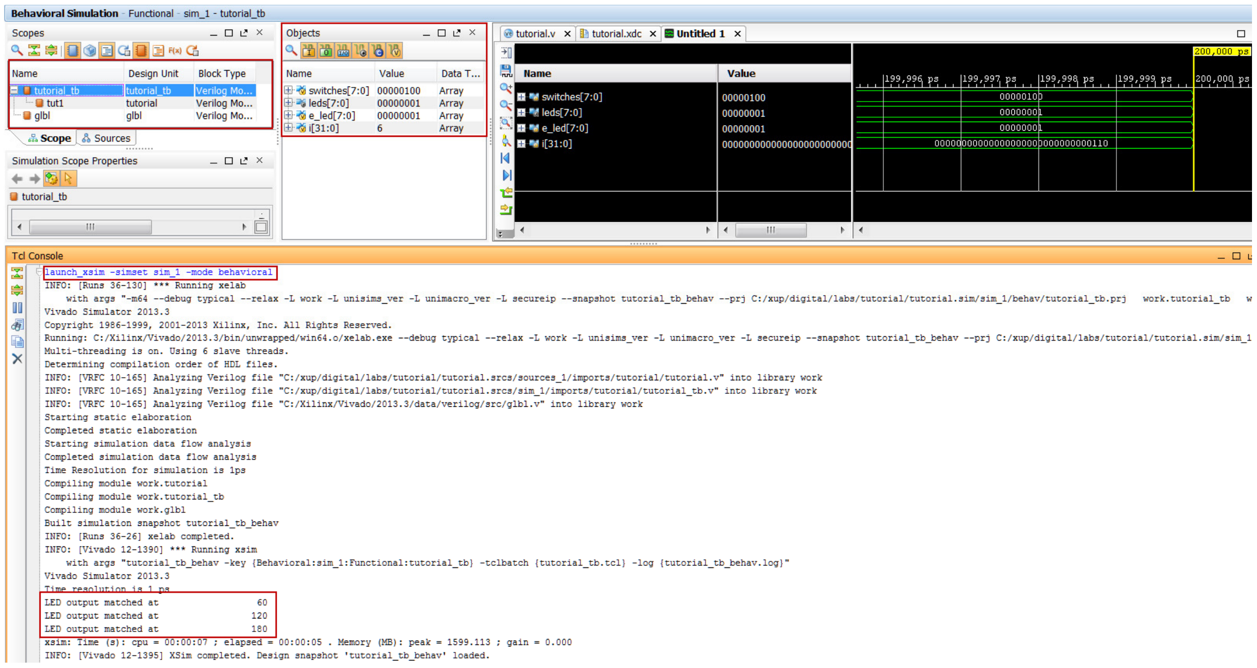
2-2-1. Select **Simulation Settings** under the *Project Manager* tasks of the *Flow Navigator* pane.

A **Project Settings** form will appear showing the **Simulation** properties form.

2-2-2. Select the **Simulation** tab, and set the **Simulation Run Time** value to 200 ns and click **OK**.

2-2-3. Click on **Run Simulation > Run Behavioral Simulation** under the *Flow Navigator* pane.

The testbench and source files will be compiled and the XSim simulator will be run (assuming no errors). You will see a simulator output similar to the one shown below.



Name	Design Unit	Block Type
tutorial_tb	tutorial_tb	Verilog Mo...
tut1	tutorial	Verilog Mo...
glbl	glbl	Verilog Mo...

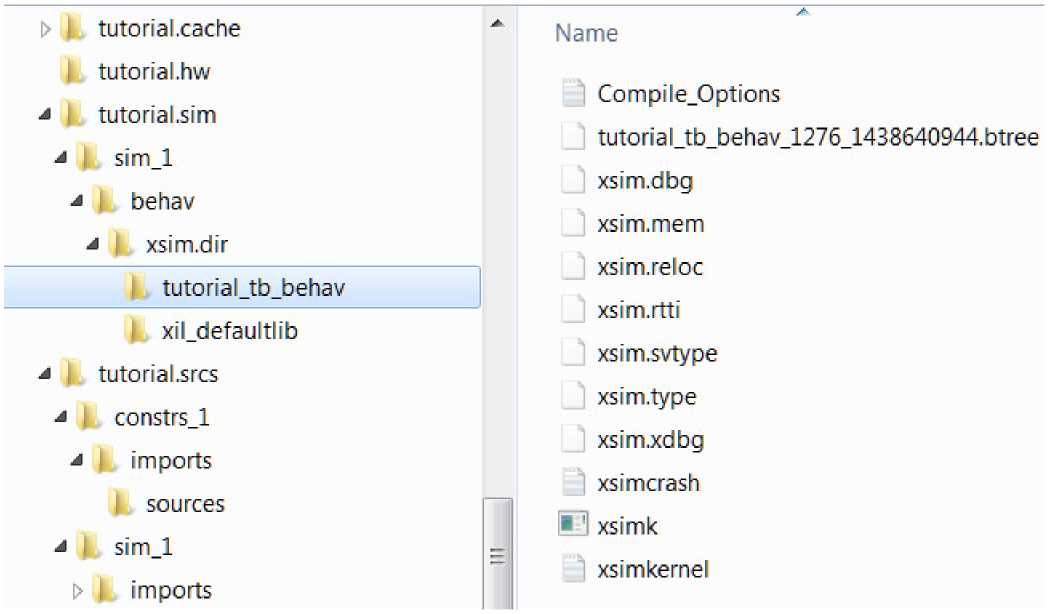
Name	Value	Data T...
switches[7:0]	00000100	Array
leds[7:0]	00000001	Array
e_led[7:0]	00000001	Array
i[31:0]	6	Array

```
launch_xsim -simset sim_1 -mode behavioral
INFO: [Runs 36-130] *** Running xelab
    with args "-m64 --debug typical --relax -L work -L unisims_ver -L unimacro_ver -L secureip --snapshot tutorial_tb_behav --prj C:/xup/digital/labs/tutorial/tutorial.sim/sim_1/behav/tutorial_tb.prj  work.tutorial_tb  w
Vivado Simulator 2013.3
Copyright 1986-1999, 2001-2013 Xilinx, Inc. All Rights Reserved.
Running: C:/Xilinx/Vivado/2013.3/bin/unwrapped/win64.o/xelab.exe --debug typical --relax -L work -L unisims_ver -L unimacro_ver -L secureip --snapshot tutorial_tb_behav --prj C:/xup/digital/labs/tutorial/tutorial.sim/sim_1
Multi-threading is on. Using 6 slave threads.
Determining compilation order of HDL files.
INFO: [VRFC 10-165] Analyzing Verilog file "C:/xup/digital/labs/tutorial/tutorial.srcs/sources_1/imports/tutorial/tutorial.v" into library work
INFO: [VRFC 10-165] Analyzing Verilog file "C:/xup/digital/labs/tutorial/tutorial.srcs/sim_1/imports/tutorial/tutorial_tb.v" into library work
INFO: [VRFC 10-165] Analyzing Verilog file "C:/Xilinx/Vivado/2013.3/data/verilog/src/glbl.v" into library work
Starting static elaboration
Completed static elaboration
Starting simulation data flow analysis
Completed simulation data flow analysis
Time Resolution for simulation is 1ps
Compiling module work.tutorial
Compiling module work.tutorial_tb
Compiling module work.glbl
Built simulation snapshot tutorial_tb_behav
INFO: [Runs 36-26] xelab completed.
INFO: [Vivado 12-1390] *** Running xsim
    with args "tutorial_tb_behav -key {Behavioral:sim_1:Functional:tutorial_tb} -tclbatch {tutorial_tb.tcl} -log {tutorial_tb_behav.log}"
Vivado Simulator 2013.3
Time resolution is 1 ps
LED output matched at                60
LED output matched at               120
LED output matched at               180
xsim: Time (s): cpu = 00:00:07 ; elapsed = 00:00:05 . Memory (MB): peak = 1599.113 ; gain = 0.000
INFO: [Vivado 12-1395] XSim completed. Design snapshot 'tutorial_tb_behav' loaded.
```

Figure 19. Simulator output

You will see four main views: (i) *Scopes,* where the testbench hierarchy as well as glbl instances are displayed, (ii) *Objects,* where top-level signals are displayed, (iii) the waveform window, and (iv) *Tcl Console* where the simulation activities are displayed. Notice that since the testbench used is self-checking, the results are displayed as the simulation is run.

Notice that the **tutorial.sim** directory is created under the **tutorial** directory, along with several lower-level directories.



```
tutorial.cache
tutorial.hw
tutorial.sim
  sim_1
    behav
      xsim.dir
        tutorial_tb_behav
        xil_defaultlib
tutorial.srcs
  constrs_1
    imports
      sources
  sim_1
    imports
```

```
Name
Compile_Options
tutorial_tb_behav_1276_1438640944.btree
xsim.dbg
xsim.mem
xsim.reloc
xsim.rtti
xsim.svtype
xsim.type
xsim.xdbg
xsimcrash
xsimk
xsimkernel
```

Figure 20. Directory structure after running behavioral simulation

Please refer to the following URL:

http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug900-vivado-logic-simulation.pdf to learn more about the Vivado simulator.

2-2-4. Click on the *Zoom Fit* button ( ) located left of the waveform window to see the entire waveform.

Notice that the output changes when the input changes.

You can also float the simulation waveform window by clicking on the Float button on the upper right hand side of the view. This will allow you to have a wider window to view the simulation waveforms. To reintegrate the floating window back into the GUI, simply click on the Dock Window button.

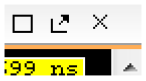


Figure 21. Float Button

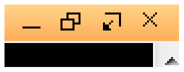


Figure 22. Dock Window Button

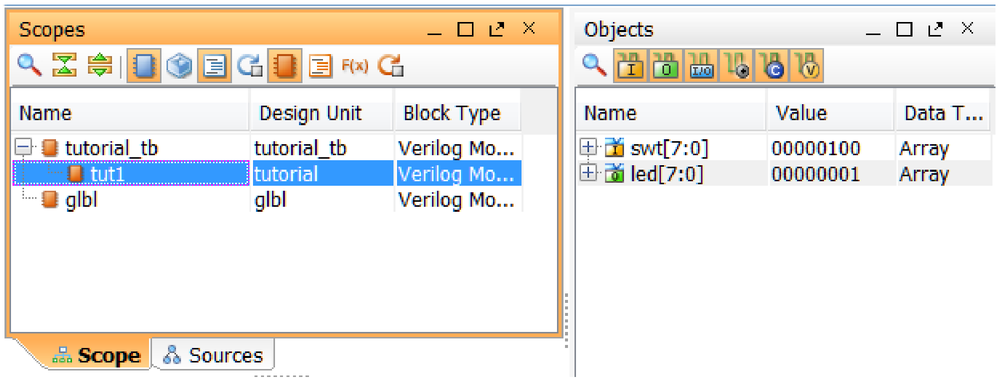
2-3. Change display format if desired.

2-3-1. Select **i[31:0]** in the waveform window, right-click, select *Radix*, and then select *Unsigned Decimal* to view the for-loop index in *integer* form. Similarly, change the radix of **switches[7:0]** to *Hexadecimal*. Leave the **leds[7:0]** and **e_led[7:0]** radix to *binary* as we want to see each output bit.

2-4. Add more signals to monitor lower-level signals and continue to run the simulation for 500 ns.

2-4-1. Expand the **tutorial_tb** instance, if necessary, in the *Scopes* window and select the **tut1** instance.

The swt[7:0] and led[7:0] signals will be displayed in the *Objects* window.



Name	Design Unit	Block Type
tutorial_tb	tutorial_tb	Verilog Mo...
tut1	tutorial	Verilog Mo...
glbl	glbl	Verilog Mo...

Name	Value	Data T...
swt[7:0]	00000100	Array
led[7:0]	00000001	Array

Figure 23. Selecting lower-level signals

2-4-2. Select **swt[7:0]** and **led[7:0]** and drag them into the waveform window to monitor those lower-level signals. You can also add the signals by right-clicking and selecting **Add to Wave Window**.

2-4-3. On the simulator tool buttons ribbon bar, type 500 in the time window, click on the drop-down button of the units field and select ns, and click on the ( ) button.

The simulation will run for an additional 500 ns.

2-4-4. Click on the *Zoom Fit* button and observe the output.

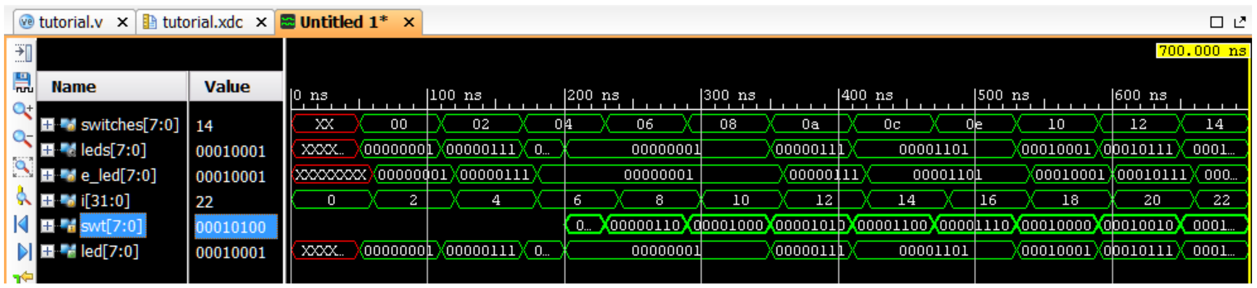


Figure 24. Running simulation for additional 500 ns

2-4-5. Close the simulator by selecting **File > Close Simulation**.

2-4-6. Click **OK** and then click **Discard** to close it without saving the waveform.

Synthesize the Design Step 3

3-1. Synthesize the design with the Vivado synthesis tool and analyze the Project Summary output.

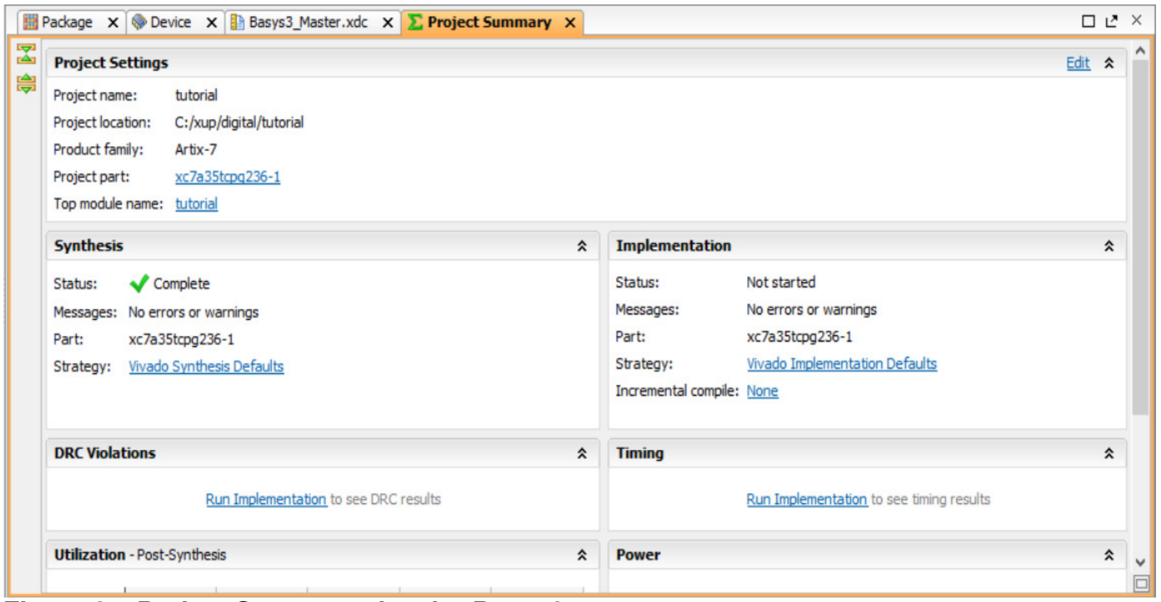
3-1-1. Click on **Run Synthesis** under the *Synthesis* tasks of the *Flow Navigator* pane.

The synthesis process will be run on the tutorial.v file (and all its hierarchical files if they exist). When the process is completed a *Synthesis Completed* dialog box with three options will be displayed.

3-1-2. Select the *Open Synthesized Design* option and click **OK** as we want to look at the synthesis output before progressing to the implementation stage.

Click **Yes** to close the elaborated design if the dialog box is displayed.

3-1-3. Select the **Project Summary** Σ tab (Select default layout if the tab is not visible) and understand the various windows.


Package | Device | Basys3_Master.xdc | Project Summary

Project Settings

Project name: tutorial
Project location: C:/xup/digital/tutorial
Product family: Artix-7
Project part: xc7a35tcpg236-1
Top module name: tutorial

Synthesis

Status: Complete
Messages: No errors or warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Synthesis Defaults

Implementation

Status: Not started
Messages: No errors or warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

DRC Violations

Run Implementation to see DRC results

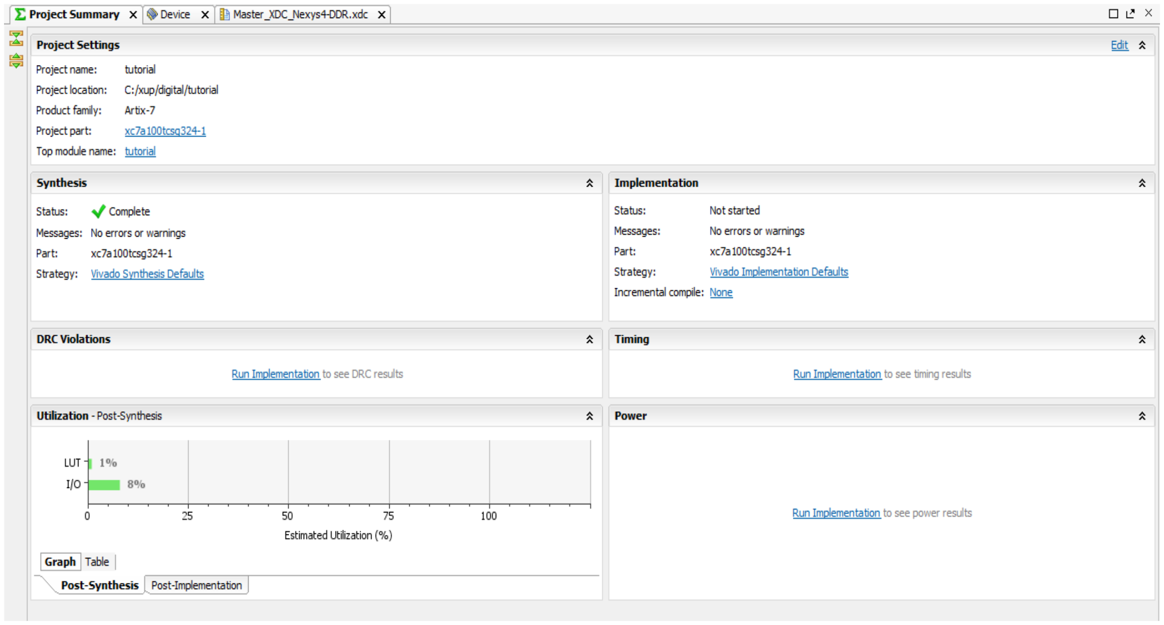
Timing

Run Implementation to see timing results

Utilization - Post-Synthesis

Power

Figure 25. Project Summary view for Basys3


Project Summary | Device | Master_XDC_Nexys4-DDR.xdc

Project Settings

Project name: tutorial
Project location: C:/xup/digital/tutorial
Product family: Artix-7
Project part: xc7a100tcsg324-1
Top module name: tutorial

Synthesis

Status: Complete
Messages: No errors or warnings
Part: xc7a100tcsg324-1
Strategy: Vivado Synthesis Defaults

Implementation

Status: Not started
Messages: No errors or warnings
Part: xc7a100tcsg324-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

DRC Violations

Run Implementation to see DRC results

Timing

Run Implementation to see timing results

Utilization - Post-Synthesis

Power

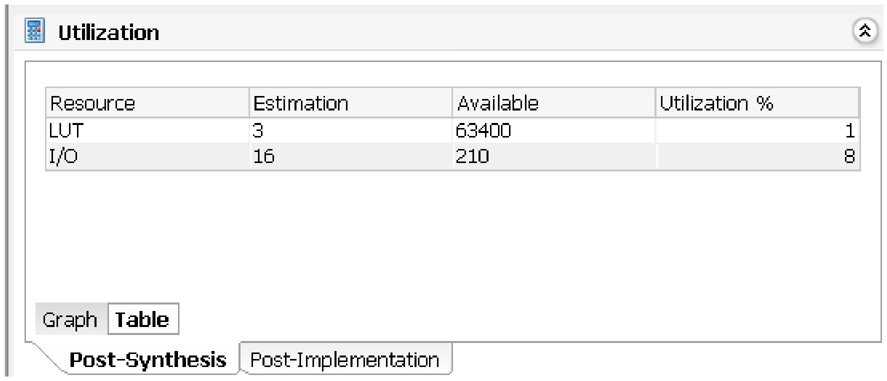
Run Implementation to see power results

Figure 25. Project Summary view for Nexys4 DDR

Click on the various links to see what information they provide and which allows you to change the synthesis settings.

3-1-4. Click on the **Table** tab in the **Project Summary** tab.

Notice that there are an estimated three LUTs and 16 IOs (8 input and 8 output) that are used.



Resource	Estimation	Available	Utilization %
LUT	3	63400	1
I/O	16	210	8

Figure 26. Resource utilization estimation summary

3-1-5. Click on **Schematic** under the *Open Synthesized Design* tasks of *Synthesis* tasks of the *Flow Navigator* pane to view the synthesized design in a schematic view.

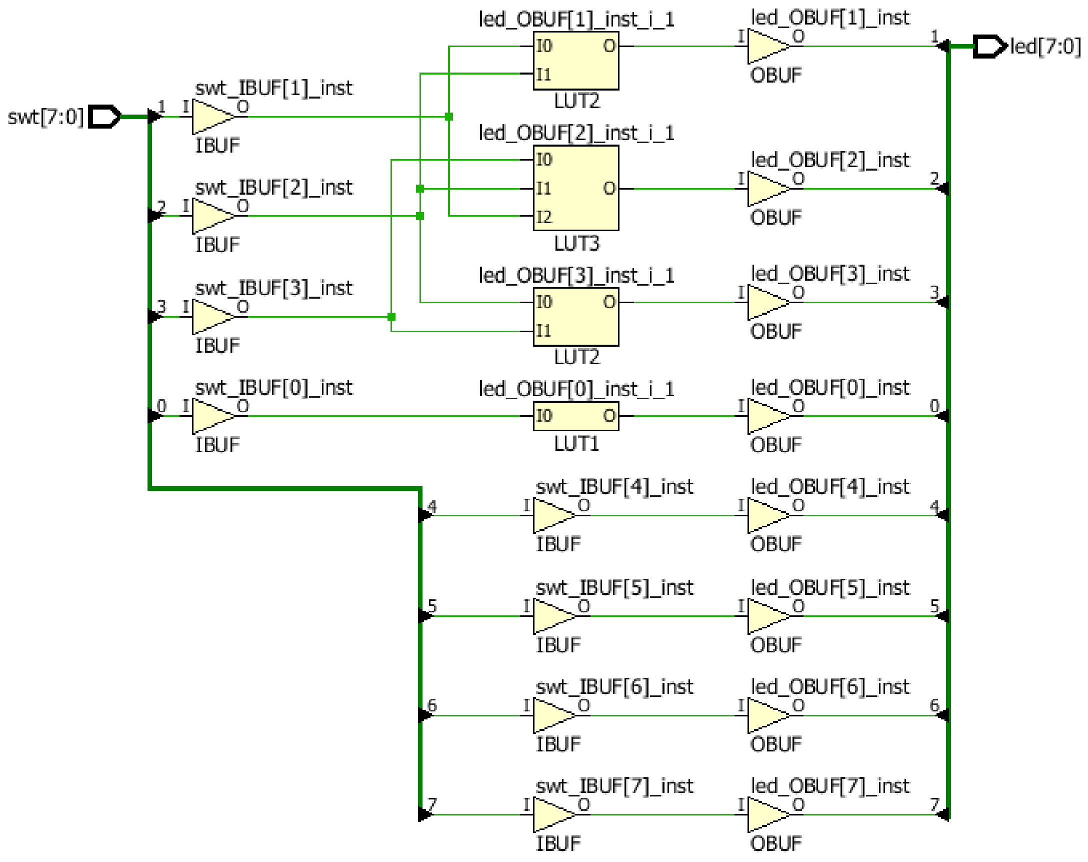


Figure 27. Synthesized design's schematic view

Notice that IBUF and OBUF are automatically instantiated (added) to the design as the input and output are buffered. The logical gates are implemented in LUTs (1 input is listed as LUT1, 2 input is listed as LUT2, and 3 input is listed as LUT3). Four gates in RTL analysis output is mapped into four LUTs in the synthesized output.

Using the Windows Explorer, verify that **tutorial.runs** directory is created under **tutorial**. Under the **runs** directory, **synth_1** directory is created which holds several temporary sub-directories.

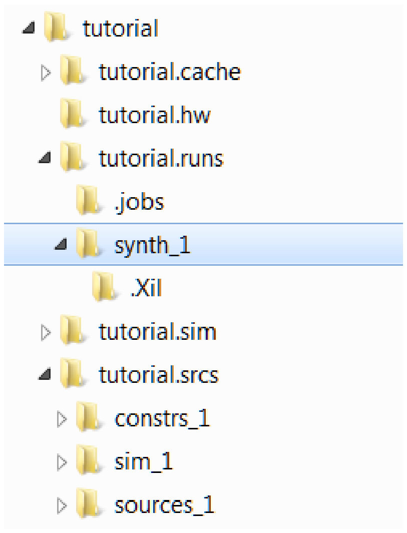


Figure 28. Directory structure after synthesizing the design

Implement the Design Step 4

4-1. Implement the design with the Vivado Defaults Implementation settings and analyze the Project Summary output.

4-1-1. Click on **Run Implementation** under the *Implementation* tasks of the *Flow Navigator* pane.

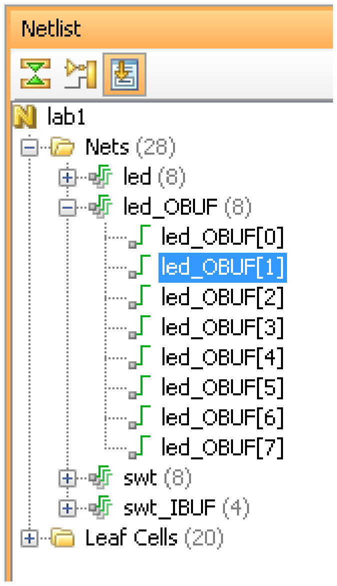
The implementation process will be run on the synthesis output files. When the process is completed an *Implementation Completed* dialog box with three options will be displayed.

4-1-2. Select **Open implemented design** and click **OK** as we want to look at the implemented design in a Device view tab.

4-1-3. Click **Yes** to close the synthesized design.

The implemented design will be opened.

4-1-4. In the *Netlist* pane, select one of the nets (e.g. led_OBUF[1]) and notice that the net displayed in the X0Y0 clock region for Basys3 and the X0Y1 and X0Y2 clock regions for Nexys4 DDR in the Device view tab (you may have to zoom in to see it).



Netlist

lab1
Nets (28)
led (8)
led_OBUF (8)
led_OBUF[0]
led_OBUF[1]
led_OBUF[2]
led_OBUF[3]
led_OBUF[4]
led_OBUF[5]
led_OBUF[6]
led_OBUF[7]
swt (8)
swt_IBUF (4)
Leaf Cells (20)

Figure 29. Selecting a net

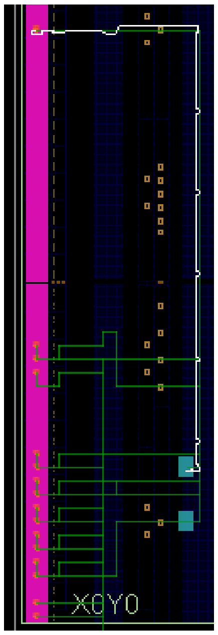


Figure 29. Viewing implemented design for Basys3

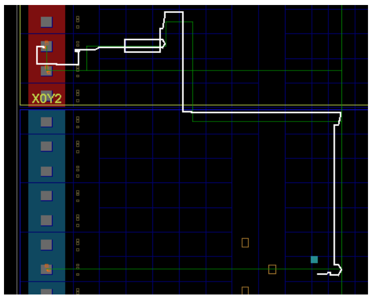
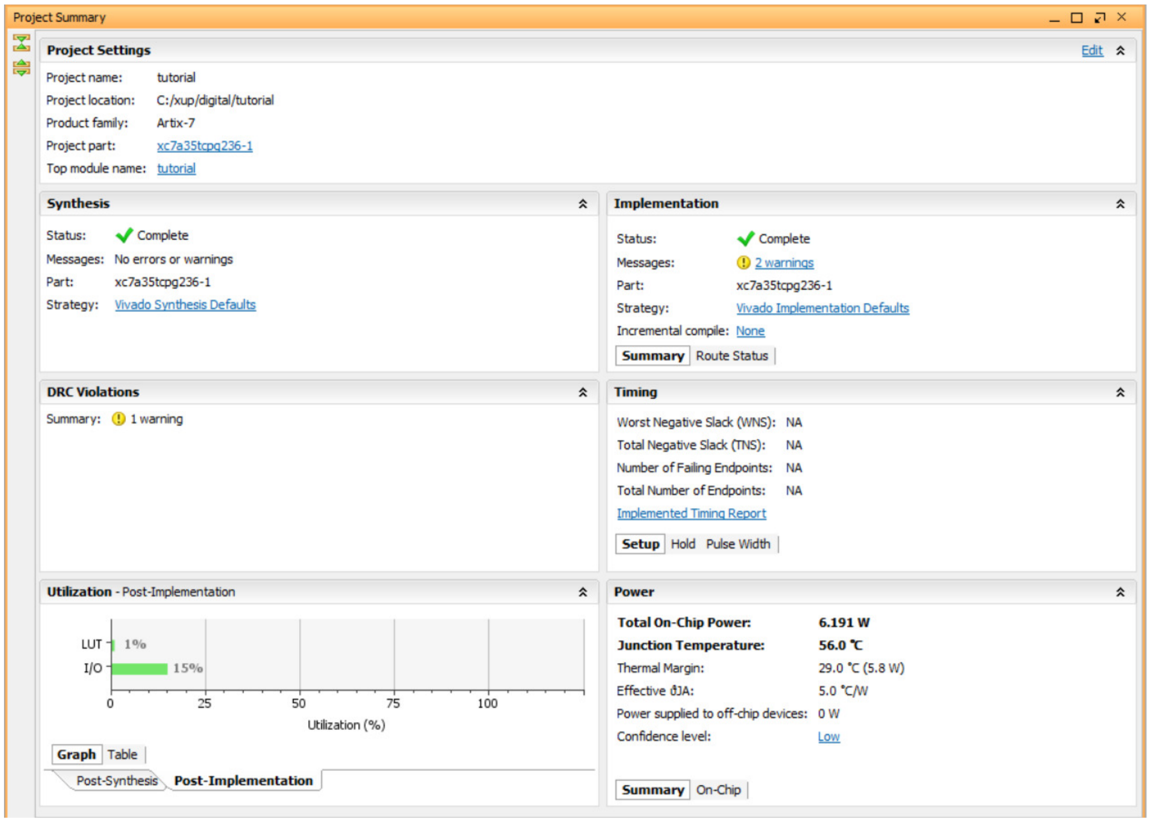


Figure 29. Viewing implemented design for Nexys4 DDR

4-1-5. Close the implemented design view by selecting **File > Close Implemented Design**, and select the **Project Summary** tab (you may have to change to the Default Layout view) and observe the results.

Notice that the actual resource utilization is three LUTs and 16 IOs. Also, it indicates that no timing constraints were defined for this design (since the design is combinatorial).Select the **Post-implementation** tabs under the *Timing* and *Utilization* windows.


Project Summary

Project Settings
Project name: tutorial
Project location: C:/xup/digital/tutorial
Product family: Artix-7
Project part: xc7a35tcpg236-1
Top module name: tutorial

Synthesis
Status: ✓ Complete
Messages: No errors or warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Synthesis Defaults

Implementation
Status: ✓ Complete
Messages: 2 warnings
Part: xc7a35tcpg236-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

DRC Violations
Summary: 1 warning

Timing
Worst Negative Slack (WNS): NA
Total Negative Slack (TNS): NA
Number of Failing Endpoints: NA
Total Number of Endpoints: NA
Implemented Timing Report

Utilization - Post-Implementation
LUT 1%
I/O 15%

Power
Total On-Chip Power: 6.191 W
Junction Temperature: 56.0 °C
Thermal Margin: 29.0 °C (5.8 W)
Effective θJA: 5.0 °C/W
Power supplied to off-chip devices: 0 W
Confidence level: Low

Figure 30. Implementation results

Using the Windows Explorer, verify that **impl_1** directory is created at the same level as **synth_1** under the **tutorial_runs** directory. The **impl_1** directory contains several files including the report files.

4-1-6. Select the *Report Utilization* entry under the *Synthesized Design* under the Flow Navigator. The report will be displayed in the auxiliary view pane showing resources utilization. Note that since the design is combinatorial no registers are used.

Perform Timing Simulation Step 5

5-1. Run a timing simulation.

5-1-1. Select **Run Simulation > Run Post-Implementation Timing Simulation** process under the *Simulation* tasks of the *Flow Navigator* pane.

The XSim simulator will be launched using the implemented design and **tutorial_tb** as the top-level module.

Using the Windows Explorer, verify that **timing** directory is created under the **tutorial.sim > sim_1 > impl** directory. The **timing** directory contains generated files to run the timing simulation.

5-1-2. Click on the **Zoom Fit** button to see the waveform window from 0 to 200 ns.

5-1-3. Right-click at 50 ns (where the switch input is set to 0000000b) and select **Markers > Add Marker**.

5-1-4. Similarly, right-click and add a marker at around 55.000 ns where the **leds** changes.

5-1-5. You can also add a marker by clicking on the Add Marker button ( ). Click on the **Add Marker** button and left-click at around 60 ns where **e_led** changes.

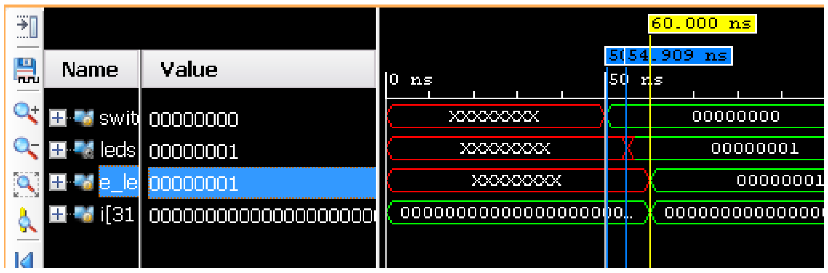


Figure 31. Timing simulation output

Notice that we monitored the expected led output at 10 ns after the input is changed (see the testbench) whereas the actual delay is about 5.000 ns.

5-1-6. Close the simulator by selecting **File > Close Simulation** without saving any changes.

Generate the Bitstream and Verify Functionality Step 6

6-1. Make sure that the power supply source jumper is set to USB. Connect the board and power it ON. Generate the bitstream, open a hardware session, and program the FPGA.

6-1-1. Make sure that the power supply source jumper is set to *USB* and the provided Micro-USB cable is connected between the board and the PC. Note that you do not need to connect the power jack and the board can be powered and configured via USB alone.

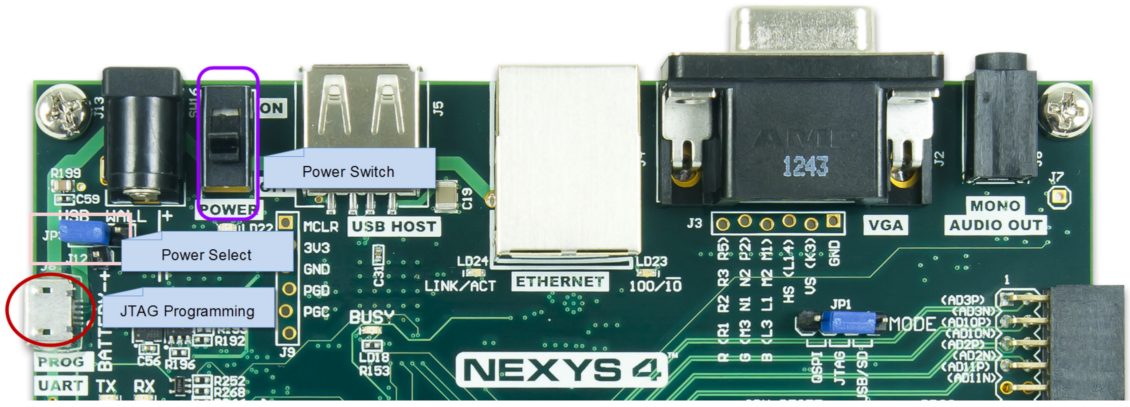


Figure 32. Board settings for Nexys4 DDR

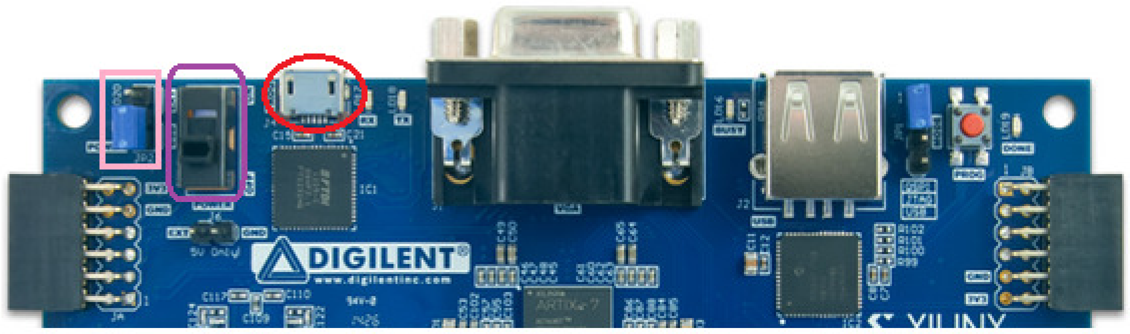


Figure 32. Boards settings for Basys3

6-1-2. Power **ON** the switch on the board.

6-1-3. Click on the **Generate Bitstream** entry under the *Program and Debug* tasks of the *Flow Navigator* pane.

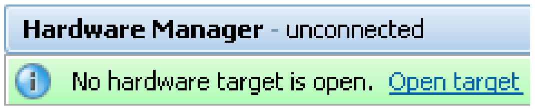
The bitstream generation process will be run on the implemented design. When the process is completed a *Bitstream Generation Completed* dialog box with three options will be displayed.

This process will have **tutorial.bit** file generated under **impl_1** directory which was generated under the **tutorial.runs** directory.

6-1-4. Select the *Open Hardware Manager* option and click **OK**.

The Hardware Manager window will open indicating "unconnected" status.

6-1-5. Click on the **Open target** link.



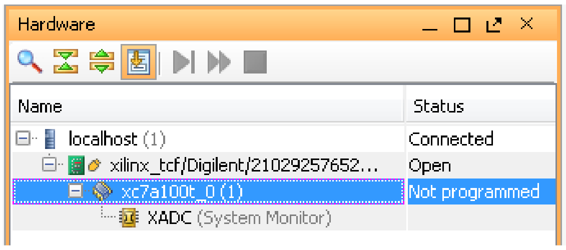
Hardware Manager - unconnected

No hardware target is open. Open target

Figure 33. Opening new hardware target

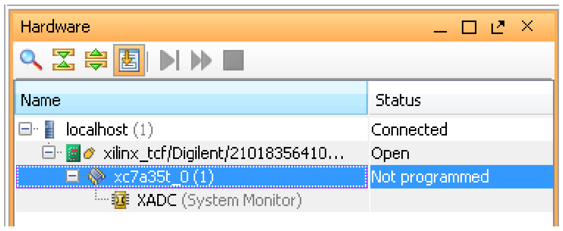
6-1-6. From the dropdown menu, click **Auto Connect.**

The Hardware Session status changes from Unconnected to the server name and the device is highlighted. Also notice that the Status indicates that it is not programmed.



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/21029257652...	Open
xc7a100t_0 (1)	Not programmed
XADC (System Monitor)	

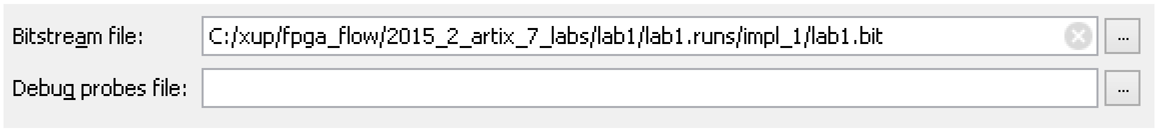
Figure 34. Opened hardware session for the Nexys4 DDR



Name	Status
localhost (1)	Connected
xilinx_tcf/Digilent/21018356410...	Open
xc7a35t_0 (1)	Not programmed
XADC (System Monitor)	

Figure 34. Opened hardware session for the Basys3

6-1-7. Select the device and verify that the lab1.bit is selected as the programming file in the General tab.



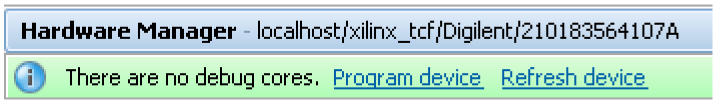
Bitstream file: C:/xup/fpga_flow/2015_2_artix_7_labs/lab1/lab1.runs/impl_1/lab1.bit

Debug probes file:

Figure 35. Programming file

6-1-8. Click on the *Program device > XC7A100T_0* or the *XC7A35T_0* link in the green information bar to program the target FPGA device.

Another way is to right click on the device and select *Program Device…*



Hardware Manager - localhost/xilinx_tcf/Digilent/210183564107A

There are no debug cores. Program device Refresh device

Figure 36. Selecting to program the FPGA

6-1-9. Click **Program** to program the FPGA.

The DONE light will light when the device is programmed. You may see some other LEDs lit depending on switch positions.

6-1-10. Verify the functionality by flipping switches and observing the output on the LEDs.

6-1-11. Close the hardware session by selecting **File > Close Hardware Manager.**

6-1-12. Click **OK** to close the session.

6-1-13. Power **OFF** the board.

6-1-14. Close the **Vivado** program by selecting **File > Exit** and click **OK**.

Conclusion

The Vivado software tool can be used to perform a complete design flow. The project was created using the supplied source files (HDL model and user constraint file). A behavioral simulation was done to verify the model functionality. The model was then synthesized, implemented, and a bitstream was generated. The timing simulation was run on the implemented design using the same testbench. The functionality was verified in hardware using the generated bitstream.